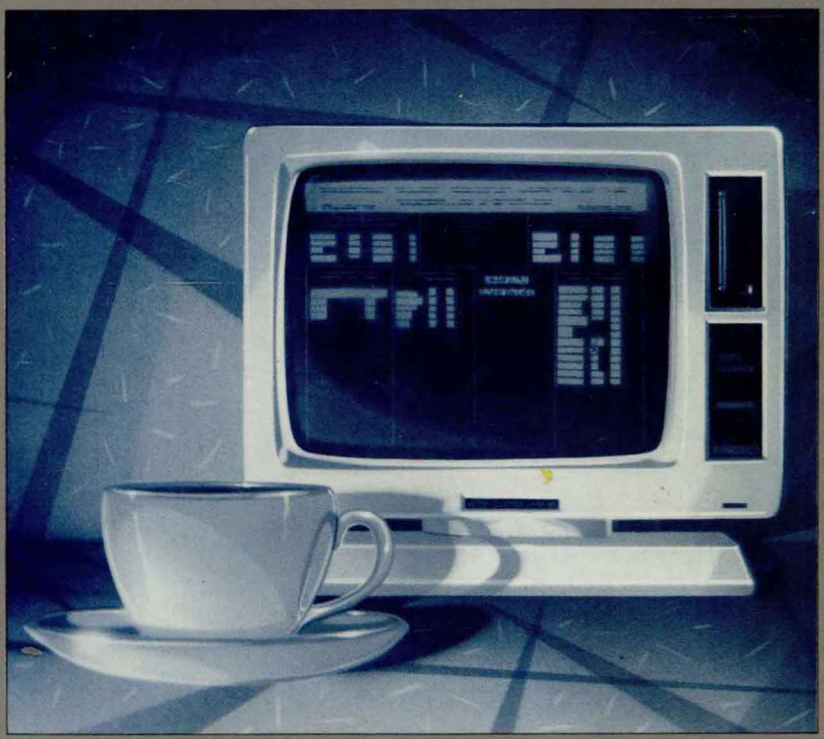
COMPUTERS AND PEOPLE SERIES

Formal Methods for Interactive Systems



Alan John Dix

Formal Methods for Interactive Systems

ALAN JOHN DIX

Department of Computer Science
University of York

ACADEMIC PRESS

Harcourt Brace Jovanovich, Publishers
London San Diego New York
Boston Sydney Tokyo Toronto

ACADEMIC PRESS LTD
24/28 Oval Road,
London NW1 7DX

United States Edition published by
ACADEMIC PRESS INC.
San Diego, California 92101-4311

A catalogue record for this book is available from the British Library

ISBN 0-12-218315-0

Printed and bound in Great Britain by the
University Press, Cambridge

Formal Methods for Interactive Systems

Preface

This book is the product of over six years of research in the Human–Computer Interaction Group at the University of York. This group includes members of the Computer Science and Psychology Departments and it has been a privilege to work in such a stimulating environment.

HCI is itself a cross-disciplinary field and this book brings together aspects of HCI with the sort of formal methods found in software engineering – at first sight, an unlikely marriage. Computers, in general, and formal methods, in particular, define strict laws of operation, yet people cannot usually be fitted into such straight-jackets. On the other hand, so many systems I have seen and used, in banks, shops, and including all that I have used to produce this book, have user-interface faults that could have been prevented by relatively simple formal analyses. So, aware of both these facts, this book contains both eulogies of the usefulness of formal techniques and warnings of their misuse.

The recurrent theme of the book is to take a class of interactive systems and to define a formal model which captures critical aspects of them. These models are then used to frame formal statements of properties relating to the systems' usability. The emphasis is on describing the systems and defining the properties. The intention is not to produce a formal notation or architecture for constructing interactive systems, but rather to develop frameworks which, among other things, guide the use of existing notations and architectures. The formal models can be used in a strictly rigorous way, as part of a formal design process, however, they also, and more importantly, act as conceptual tools to aid our understanding of the systems.

There will be few readers in this strange border country between HCI and formal methods, so I assume that most will be in one camp or the other. I hope that it will prove useful to both camps: to those working in HCI who feel the need for a more rigorous framework for aspects of their work; and also to those software engineers working within increasingly formal methodologies who recognise a need to address the people who use their systems. There may be a temptation for the latter to notice only the eulogies and the former the warnings, however, I hope that you will heed both.

Those who work in HCI, whether drawn from psychology, sociology or computing tend to share an interest in people. Is this because they are self-selecting, or because their jobs emphasise an empathy with their users? I'm not sure, but it places them in a position of responsibility within their workplace and

within the information technology community, to reflect humanity amid mechanism. Their job actually asks them to balance the law of the machine with love, mercy and care for its users.

The balance between law and mercy is never easy, and naturally involves a whole wealth of ethical, philosophical and religious issues. In the work described here, there is a clear stance, which I never deliberately took, but gradually noticed in my work. I have a disinclination to build models of the user and distrust of excessively task-oriented analyses. Instead, the models are formal models *of* the system *from* the user's viewpoint. The principles primarily address issues of observability and control. That is, they emphasise the user as being in charge of the interaction, and don't presume to determine precisely what the user will do.

Several parts of this book have appeared in modified form in previous published papers. I would like to thank the following for permission to reuse material: Cambridge University Press for "The myth of the infinitely fast machine" which appeared in *People and Computers III* (1987), "Abstract, generic models of interactive systems" in *People and Computers IV* (1988) and "Nondeterminism as a paradigm for understanding problems in the user interface" in *Formal Methods in Human-Computer Interaction* (1989); and Butterworths for "Interactive systems design and formal refinement are incompatible?" which appeared in *The Theory and Practice of Refinement*.

Finally I would like to thank my family, Fiona, Esther and Ruth, for their love which counterbalanced the law of the ever present PC (even on holiday!); and to thank God, our Father, who brought us to York and has sustained us since, and in whom law and love meet in costly but complete harmony.

CONTENTS

CHAPTER 1

Introduction

1.1 Formal methods and interactive systems

This books looks at issues in the interplay between two growth areas of computing research: formal methods and human–computer interaction. The practitioners and styles of the two camps are very different and it can be an uneasy path to tread.

1.1.1 Formal methods

As the memory of computers has increased, so also has the size and complexity of the software designed for them. Maintaining and understanding these systems has become a major task. Further, the range of tasks under direct control of computers has increased and the effects of failure, in say a space station or a nuclear power plant, have likewise increased. There may be little or no opportunity for direct control if the software malfunctions, for instance, in a fly-by-wire aircraft. Further, the costs of producing software have increased dramatically, and the possibility of maintaining code has diminished. This cumulation of factors has been termed the *software crisis* (Pressman 1982) and has led to a call for software design to become more of an engineering discipline. In particular, there is the desire for a more rigorous approach to software design, possibly including elements of mathematical formalism and having the possibility of being proved or at least partially checked.

Several varieties of formal methods have been developed for different purposes:

- *Graphical methods* – Such methods include Jackson Structured Programming (JSP) (Jackson 1983) for the design of data processing (DP) systems and various dataflow methods (Yourdon and Constantine 1978). Typically, only a portion of the required information is held in the graphs.

The rest may be informally annotated or there may be additional non-graphical notations, as is the case with JSP. Again the notation may be self standing, or be part of a larger methodology.

- *Program proof rules and semantics* – Another strand of formality concentrates on proving properties of programs or program fragments (Hoare 1969, Dijkstra 1976), implicitly defining a meaning for the language. Others search for more explicit expression of program language semantics (Stoy 1977).
- *Specification notations* – These are languages and notations specifically designed for the formal specification of software. Examples include Vienna Definition Method (VDM) (Jones 1980), Clear (Burstall and Goguen 1980) and Oxford's Z notation (Sufrin *et al.* 1985, Morgan 1985). Often these notations have explicit rules for the correct transformation of a specification towards an implementable form (Jones 1980).
- *Methodologies* – Instead, or as well as defining precisely how a design is to be specified or proved, there are many methodologies aimed at defining what should happen in the design *process*. For example, JSP, mentioned above, is part of a complete design process. These formal methodologies may incorporate parts of the design process that are beyond what could be expected of an entirely rigorous approach. These approaches may involve graphical and textual notations and may be amenable to computer verification of consistency (Stephens and Whitehead 1986).

There is an underlying assumption to much of this book that software is being developed using some formal notation of the third category. Various sections of the book propose parts of a semi-formal design methodology.

1.1.2 Interactive systems

In the early days of computing the modes of interaction with the user were severely limited by the hardware available: initially cards and switches, later teletypes. The more limited the capabilities the greater the need for effective interface design, but early users were usually experts and there was little spare processing power for frills. Even as processing power increased and the interface hardware improved there was still a strong pull from the experts, who were the major users, for power and complexity. Two major influences have pushed the computer industry towards improved user interface design:

- *Technology push* – The realisation that there was the possibility of a computer society prompted research using state-of-the-art technology into futuristic scenarios. The Xerox work on Smalltalk (Goldberg 1984) and the Star interface (Smith *et al.* 1983) are examples of this.

- *Large user base* – The plummetting cost of hardware has led to a huge growth in computers in the hands of non-computer-professionals. The personal computer boom has taken the computer out of the hands of the DP department, and the new users are not prepared for inconsistent and obscure software.

The two strands are not independent. The Xerox Star has led to the very popular Macintosh, and the WIMP (windows, mice and pop-up menus) interface has become standard in the market-place. If one were to balance the two, it is perhaps the latter strand which is really of most significance in the current high prominence of issues of human–computer interaction (HCI).

The perceived importance of HCI is evidenced by the large number of conferences dedicated to it: the CHI conferences in USA, HCI in Britain, INTERACT in Europe, and HCI International. The "man–machine" interface was also a major strand of the Alvey initiative (Alvey 1984) and (under a different name) of its successor, the IED program. HCI also has a prominent role in the European Community's Esprit program.

1.1.3 The meeting

There is a certain amount of culture shock when first bringing together the concepts of formal methods and interactive systems design. The former are largely perceived as dry and uninspiring, in line with the popular image of mathematics. Interface design is, on the other hand, a more colourful and exciting affair. Smalltalk, for instance, is not so much a programming environment as a popular culture. Also it is hard to reconcile the multi-facetedness of the user with the rigours of formal notations. Some of these problems may be to do with misunderstandings about the nature of formalism (although even I, a mathematician, find a lot of computer science formalism very dry). However, this is not a problem just between formalisms and users: mathematical and formal reasoning typically is performed by people and does therefore have a more human side. The shock really occurred when living users met dry unemotional computers and must therefore be dealt with in any branch of HCI. However, the gut reaction still exists and is a reminder of the delicate balance between the two.

No matter how strong the reaction against it, there is clearly a necessity for a blending of formal specification and human factors of interactive systems. If systems are increasingly designed using formal methods, this will inevitably affect the human interface, and if the issue isn't addressed explicitly the methods used will not be to the advantage of the interface designer. If we look again at some of the reasons for needing formal methods, large critical systems where the crisis is most in evidence clearly need an effective interface to their complexity. The penalty for not including this interface in the formal standards will be more

accidents due to human error such as at Chernobyl and Three Mile Island, and the more powerful the magnifying effect of the control system the more damaging the possible effects.

The need for more formal design is seen also in more mundane software. Many of the problems in interactive systems are with awkward boundary cases and inconsistent behaviour. These are obvious targets for a formal approach.

1.1.4 Formal approaches in HCI

There are several approaches taken to the formal development of interactive systems:

- *Psychological and soft computer science notations* – These include the layered approach of Foley and van Dam (1982), or the more cognitive and goal-oriented methods such as TAGPayne 1984. The uses of these vary, for instance improving design, predicting user response times and predicting user errors. They are not intended for combination with the formal notations of software engineering.
- *Specifying interactive systems in existing notations* – Several authors use notations intended for general software design to specify interactive systems. Examples of this include Sufrin's elegant specification of a text editor using Z (Sufrin 1982), a similar one by Ehrig and Mahr (1985) in the ACT ONE language, and no less than four specifications in a paper by Chi (1985) in which he compares different formal notations for interface specification. Sometimes it is some component of the interface that is specified rather than an entire interactive system, as is the case with the Presenter, an autonomous display manager described by Took (1986a, 1986b, 1990). Pure functional languages have also been used to specify (and implement) interactive systems. Cook (1986) describes how generic interface components can be specified by using a pure functional language and Runciman (1989) has developed the PIE model, described later in this book, in a functional framework.
- *Notations for specification* – A general-purpose notation is not necessarily best suited to specifying the user interface, and various special purpose notations have been developed for interface, and especially dialogue, design. Hekmatpour and Ince, for instance, have a separate user interface design component in their specification language EPROL (Hekmatpour and Ince 1987). Marshall (1986) has merged a graphical interface specification technique with VDM in order to obtain the best of both worlds. Alexander (1987a, 1987b) has designed an executable specification/prototyping language around CSP and functional programming.

- *Modelling of users* – Another strand of work concerns the formalisation of the user. This may take the form of complex cognitive models using techniques of artificial intelligence, such as the expert system for interface design described by Wilson *et al.* (1986). Another proposal is *programmable user models*, an architecture for which programs can be writen that simulate the use of an interface. The approach is advocated by Young *et al.* (1989) with the intention of studying user cognitive processes. It has also been advocated by Runciman and Hammond (1986) and Kiss and Pinder (1986) with the aim of using the complexity of the user programs to assess the complexity of the interface.
- *Architectural models* – Any specification or piece of software has some architectural design, and specific user interface architectures have been designed with the aim of rationalising the construction of interactive systems and improving component reuse. These may be structuring techniques for existing languages such as PAC (Coutaz 1987) (Presentation–Abstraction–Control) an hierarchical agent-oriented description technique, or the MVC (Model–View–Control) paradigm used in many Smalltalk interfaces; or may be part of an overall system as is the case with UIMS (Pfaff 1985) (User Interface Management Systems). Architectural techniques are often combined with notations for dialogue design and (more rarely) interface semantics. Production rules, for example, are frequently used as the dialogue formalism in UIMS. On the other hand, interface design notations may implicitly or explicitly encourage particular architectural styles.

More extensive reviews of these different areas can be found in Alexander's thesis (Alexander 1987c) and in a report on formal interface notations and methods produced collaboratively between York and PRG Oxford. (Abowd *et al.* 1989) A recent collection of essays on the subject of formal methods in HCI edited by Harrison and Thimbleby (1989) contains papers in most of the above categories.

An additional category has become characteristic of the "York approach" to HCI, which is the main subject of this book:

- *Formal, abstract models of interaction* – These are formal descriptions of the *external* behaviour of systems. They are not models of specific systems, but each covers a *class* of interactive systems, enabling us to reason about and discuss interactive systems in the abstract. As well as a large body of work originating in York, the approach has been taken up by Anderson (1985, 1986), who uses a blend of formal language and denotational semantics to describe interactive systems, and by Sufrin and He (1989), who cast in Z, a model similar to the *PIE* model presented in Chapter 2.

We can lay out the formal approaches to interactive systems design in a matrix classified by concreteness and by generality (*fig.* 1.1). The concreteness axis distinguishes between the internal workings of the systems and the specification of their external behaviour. The former are more useful for producing systems, the latter for reasoning about them. The generality of a method may lie between those which can be realised only in the context of a specific system and those that have some existence over a class. Laid out like this, it is obvious that abstract models fill a crucial gap.

	generality	
concreteness	specific	generic
specification	notations for specification *task and goal descriptions*	abstract models
implementation	prototypes of the actual system *programmable user models*	architectural models *cognitive architectures*

figure 1.1 *formal methods matrix*

In drawing up the matrix (and making my point!) I have rather overplayed the gap filled by abstract models. Specifications of particular systems may be deliberately vague in places, and thus begin to encroach on the generality barrier. Similarly, architectural models, although aimed at implementation, may be given a suitable form for us to use for specification and reasoning, and hence begin to move up towards the domain of formal models. Cockton's work (Cockton 1986) is a good example of this. He uses a description technique drawing on an analysis of UIMS. The notation is used to express properties of interface separability and comes close in spirit to the idea of an abstract interface model.

From the other side, the abstract models in this book are supplemented by examples of specifications of parts of actual systems, hence bridging the generality barrier from their side. Also, especially in Chapters 8 and 9, there is a movement towards more architectural descriptions, that is, a movement towards concreteness. Abowd (Abowd 1990) has produced a notation which attempts to sit in this middle ground between the formal models of this book and architectural models. It is, of course, no good describing useful properties of systems in a highly abstract manner, if these cannot be related to more concrete and specific situations, and thus these areas where the various techniques overlap are most important.

The more psychologically based formalisms sit rather uneasily in the matrix, but I have included them as, to the extent that they do fit the classifications, a similar gap is seen on their side. Now the abstract models we will deal with are primarily descriptions of the system *from* the user's point of view. (But definitely *not* in the language a typical user would use!) They do have then an implicit abstract, generic model of the user, purely because of the perspective from which they are drawn. It is though a rather simple model, and a more explicit model might be useful. On the other hand, I find myself feeling rather uneasy about the idea of producing generic models of users: individuality is far too precious.

1.2 Abstract models

We have seen that abstract models fill a niche in the range of available HCI formalisms, but we also need to be sure that it is a gap worth filling. We shall take a quick look at why we need abstract models, and at the philosophy behind them.

1.2.1 Principled design

There are many principles for the design of interactive systems. Some are very specific (e.g. "error messages should be in red") and others cover more general properties (e.g. "what you see is what you get"). Hansen (1984) talks about using user engineering principles in the design of a syntax-directed editor Emily. Bornat and Thimbleby (1986) describe the use of principles in the design of the display editor ded.

Thimbleby (1984) introduced the concept of *generative user engineering principles* (GUEPS). These are principles having several properties:

- They apply to a large class of different systems: that is, they are *generic*.
- They can be given both an informal *colloquial* statement and also a *formal* statement.
- They can be used to constrain the design of a given system: that is, they *generate* the design.

The last requirement can be met at an informal level using the colloquial statement – as was the case with the development of ded. While not superceding this, it seems that at least some of the generative effect should be obtained using the formal statements. The authors cited above who have specified particular interactive systems have proved certain properties of their systems, by stating the properties they require in terms of the particular specification and then proving